Implementing optimization with process simulation

LUC LAPERRIÈRE AND LARRY WASIK

ABSTRACT: For the past few years, the pulp and paper industry has faced the difficult challenge of doing more with less, and commercial simulation has become a valuable tool for this purpose. Doing "more" often means optimizing the process, which in turn means finding combinations of process parameter—values that yield optimum measures of process performance, with the least sensitivity to process disturbances. In the context of most current commercial simulators, this view of optimization presents two important problems. First, the category of process variables often used as qualitative metrics in process performance evaluation and optimization are usually absent from commercial simulator models, i.e. paper strength or pulp color do not lend themselves to mass and energy balances. Second, optimization is often performed manually by using combinations of trial-and-error variables. Most simulators do not make use of mathematically robust optimization procedures that search for the optimal combination of process variables automatically and systematically. In a previous article, we presented a potential solution to the qualitative metrics problem by developing and implementing a neural network-based module that can perform independently of the classic heat and material balance of process variables [5]. In this paper, we tackle the problem of optimizing some important process quality metrics appearing in new models by systematically searching the "space" of process parameter values that yield their maximum or minimum. We used a simulated annealing process, described in this paper, which demonstrates the well-known downhill simplex method for this purpose. We also include and discuss an example simulation that uses a newly developed optimization module.

Application: Mills can benefit from incorporating this type of optimizer into process simulation models to help achieve specific target goals, such as improved product quality and lower production costs.

Process simulation is an optimization tool. A literature search reveals many research projects concerned with finding an appropriate process model, simulating it, and then using it for optimization purposes [1-4]. However, the research rarely uses the appropriate tools to deal with the optimization and lacks a rigorous definition of what constitutes optimization. There are at least two reasons for this:

On the practical side, optimization should be performed on essential process and quality variables. Such important variables often are simply absent from the simulated models. At best, we can correlate *a posteriori* certain simulated variables with certain essential characteristics. What is lacking here is the ability of the simulator to represent such important variables natively in its mathematical models. We dealt with this first obstacle to true process optimization in an earlier study by using neural models independent of the type of input-output variables being functionally related [5].

On the mathematical side, optimization should be performed using tools that systematically, exhaustively, and efficiently search for combinations of independent variable values that lead to optimal values of the dependent variable(s) [6]. This is very different than performing optimization by manually testing some variable values until a satisfactory dependant variable value is obtained, even when such values are native to the simulated models. What is required here is additional mathematics on top of the model in order to perform true process optimization.

OPTIMIZATION PROCEDURES

The optimization module allows you to find the combinations of values of independent variables that will yield a minimum value of the dependant variable. Technically, it is a multi-variable, non-linear, constrained optimizer. *Multi-variable* means it can accept any number of independent variables. These variables can be any variable used in a simulation. *Non-linear* means the relationship between the dependent and the independent variables can be non-linear. *Constrained* means the search space can be restricted by specifying minimum and maximum values on each independent variable. This corresponds to the definition of hyperplanes in a multi-dimensional space, which the optimizer will not cross. The following is a brief description of the algorithms we chose.

Downhill simplex method

For the purposes of this discussion, let us define some useful terms:

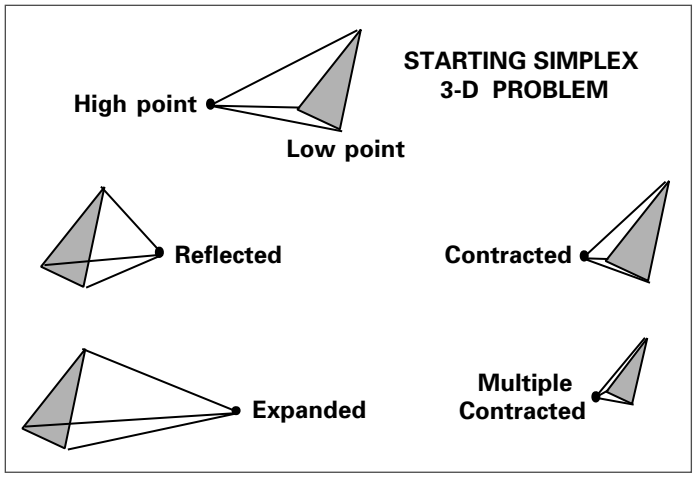
Coordinate—an independent variable in multidimensional space

Input point—a set of coordinates in multidimensional space;

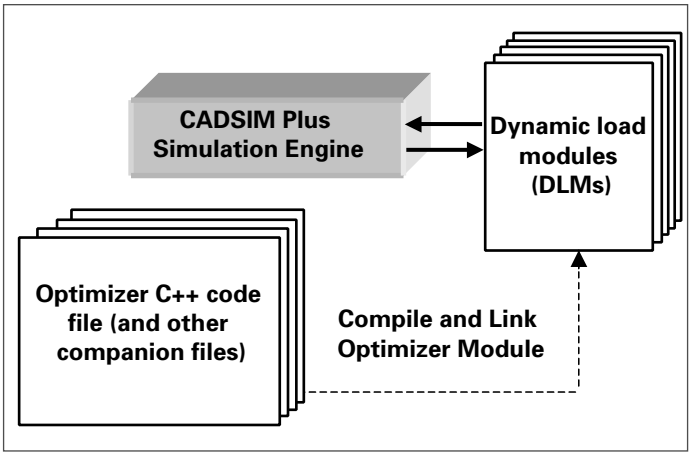
Output point or **function value**—the value of the dependent variable resulting from a particular input point

Simplex—a geometrical figure whose vertices consist of $n+1$ output points for an n dimensional coordinate space, with all vertices connected by segments. For example, in two dimensions a simplex takes the form of a planar triangle with three line-connected vertices.

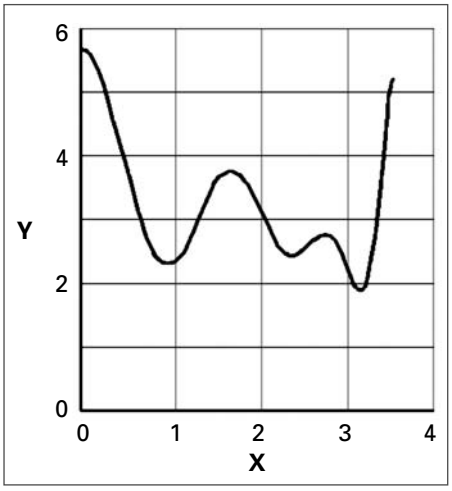
With these definitions in mind, starting with an initial simplex, the standard simplex method amounts to replacing a simplex point during each iteration. That results in a change of its topology such that it finds its way toward a local minimum of the function to be optimized. Points change through four types of moves:



1. Possible downhill simplex moves.



2. Integration of an optimizer module into a commercial simulation software package.



3. A 14th order polynomial with three minimums.

1. *reflecting* the worst point in the simplex,
2. *expanding* the worst point in the simplex,
3. *contracting* the worst point in the simplex toward the best one, or
4. *multiple contracting* all points in the simplex (with the exception of the best point) toward the best point (see **Fig. 1**).

By using a carefully designed sequence of such moves that depend on how big an improvement the newly generated simplex point represents when compared to the worst point, this algorithm converges toward a local minimum. In other words, it will contract the simplex toward its best point, possibly replacing its current best simplex point along the way, when some move generates an even better point. This procedure is not guaranteed to find a global minimum of the function to be optimized [6-8].

This simplex method does not require an explicit or symbolic encoding of the function to be optimized within the optimizer module itself. All functional relationships among simulated variables remain within the existing simulator. Therefore, the optimizer simply polls the simulator during each iteration for an output point and uses this newly returned function value to generate the new set of coordinates corresponding to the chosen simplex move. This method lets the optimizer module concentrate on its fundamental purpose (i.e. converging toward a function minimum using simplex moves). It also lets the simulator engine do what it is best at—computing functional relationships among simulated variables.

Modified simulated annealing simplex method

Research in artificial intelligence has lead to the formulation of a general and powerful tool, namely simulated annealing [9], that can be exploited in many different contexts, including multivariable function optimization. In fact, you can direct the optimizer to use the downhill simplex method described above, or a simulated annealing version of it.

The simulated annealing version of the simplex method is a rather original approach that amounts to modifying the true value of the returned function value and giving it a random thermal fluctuation proportional to some current temperature value [7]. If the temperature is initially high enough and does not decrease too rapidly, the simplex will very likely shrink into a region of the search space where the overall minimal point lies, thus avoiding the local minimum traps that can catch the standard simplex.

As an analogy, consider the simplex as a bird searching for the deepest valley in which to land. The bird overlooking a landscape with many hills and valleys of different heights and depths corresponds to the function topology in multidimensional space. With the standard simplex, if the bird is not high enough (which corresponds to a bad set of initial points in the simplex), then it cannot see over the hills surrounding it. It will necessarily land in the deepest valley within its sight. The simulated annealing method amounts to giving the bird the energy to rise high enough to have a global overview of the landscape. Reducing the annealing temperature amounts to lowering the bird's altitude in regions of the landscape where the deepest valley possibly lies. With proper parameters, the simulated annealing method systematically finds better function values than its standard simplex counterpart.

Inherent in the simulated annealing method are the notions of initial temperature and temperature decrease rate. Our implementation of this algorithm provides an automatic initial temperature proportional to the difference between the highest and lowest output point in the random initial simplex. We also fix the rate of temperature decrease such that a new temperature is at most 10% lower than the previous one. Temperature changes occur under two circumstances. First, a temperature change will necessarily occur when the simulation has performed 100 iterations at the same tem-

perature. Second, a temperature change will occur before 100 iterations if a fractional convergence of the simplex output points occurs. The user determines the fractional convergence, which is compared to the value returned by the equation:

$$ftol = \frac{2.0 \cdot \left| y_{high} - y_{low} \right|}{\left(\left| y_{high} \right| + \left| y_{low} \right| \right)} \quad (1)$$

where:

y_{high} is current high point in the simplex
y_{low} is current low point in the simplex.

Using high values of the fractional convergence (say 1) will cause frequent temperature changes, therefore speeding convergence toward a minimum, but decreasing the probability for the simplex to shrink into the true optimal region.

Integration with a commercial simulator

As seen in **Fig. 2**, implementation of the optimizer is achieved with the use of a programmatic interface enabling the design of custom computational modules to be executed within the commercial software. These modules were programmed in C++. The result of compilation and linking is an executable file that is added to the list of the software's Dynamic Load Modules (DLMs) [10]. The developed optimization module is not limited to performing its calculations in a simulation of its own, but can interact with any other modules or variables of any existing simulation, making it very generic and flexible in its use.

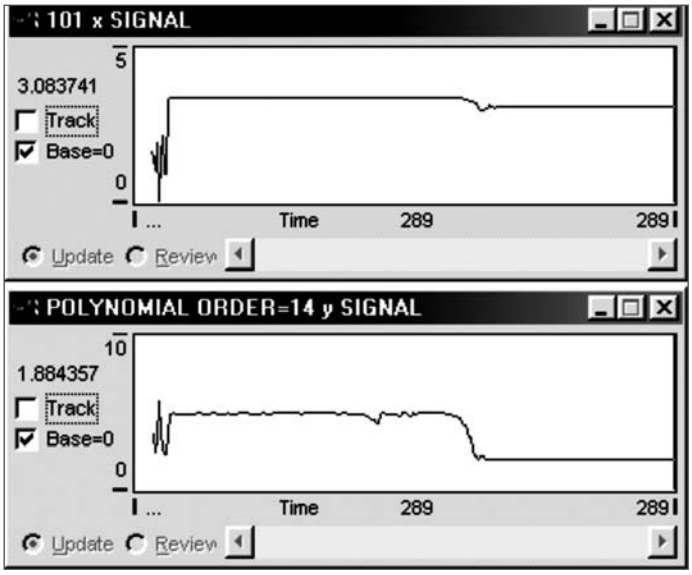
Testing

We used a 14th order polynomial in the range (0, 3.4) for "x." For this range, there are three minimums for "y," represented by the points (0.94, 2.31), (2.31, 2.46), and (3.08, 1.88). **Figure 3** shows the portion of the curve with the three minimums.

We defined the two constraints x<3.54 and x>0 for this problem, in order to restrict the search to a region where at least three minimums lie. We ran the optimizer with the standard simplex method first. Any of the three minimum is returned in this mode, depending on the random conditions for the initial points in the simplex.

Running the same problem with the simulated annealing version finds the global optimum of this search region about 93% of the time. When increasing the fractional convergence factor, the convergence becomes faster but the global minimum is less often found. **Figure 4** shows a convergence to the global minimum using simulated annealing for this problem.

When using the simplex method (or its simulated annealing version) any point being replaced should never yield a degenerate simplex (for a 2-D problem with a three-points simplex, a degenerate case would be points perfectly aligned on a straight line). This situation is likely to occur for constrained optimization where all points might get aligned at some hyperplane during the search. The initial flat portion for "x" in Fig. 4 shows exactly this situation where the simplex points hit the constraint plane at x = 3.54. To avoid a degenerate simplex, the algorithm adds a small random value to the function value at that point, so any two points blocked at a hyperplane will



4. Finding the global minimum after bouncing on a hyperplane using the simulated annealing method.

"bounce" on it and never share truly common coordinates. We tested this strategy and found that it worked well even if the optimal value lay at some hyperplane. For the case at hand, we see that after a few iterations of such "bouncing," the simplex regains shape and falls into the global minimum.

EXAMPLE APPLICATION

Figure 5 shows an example of a simulation where we trained a neural network module to model the mechanical pulping of *Acer saccharum* (maple tree). We conducted experiments at the Pulp and Paper Research Center at Université du Québec à Trois-Rivières in Canada using a Sunds Defibrator CD-300 pilot plant. The experiment included four independent variables (temperature, NaOH, Na_2SO_3, and plate gap) and five measured dependent variables (refining energy, Canadian Standard Freeness (CSF), breaking length, tear index, and ISO brightness). The model was successfully tested and we believe it represents a reasonably good generalization of the functional relationship between these variables [5]. The goal was to try to find optimal combinations of the independent variables in order to obtain the best possible combination of some dependent variables. Figure 5 also shows the optimizer module developed for this purpose (at bottom).

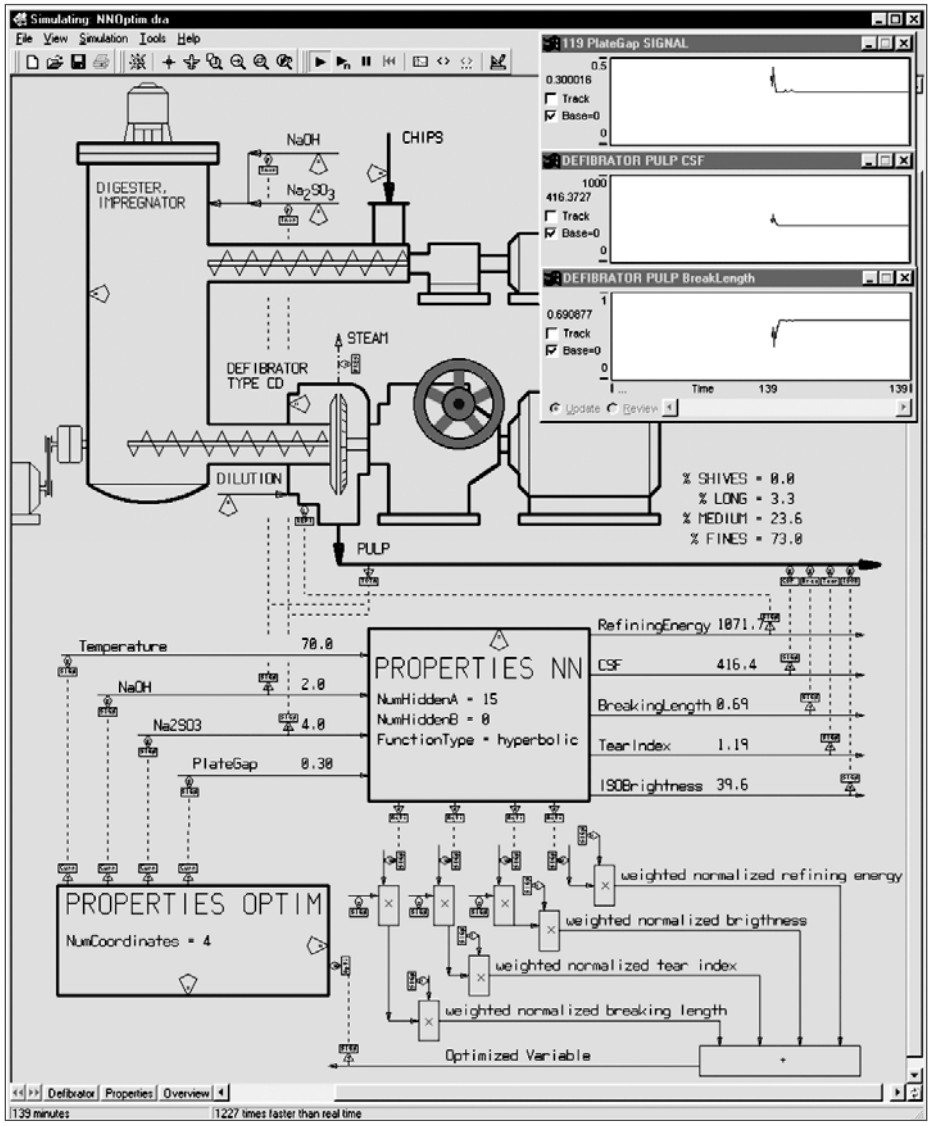
Multiple dependent variables

The developed module can accept only one dependent variable to optimize at a time. Since the neural net models the functional relationship between four inputs and five outputs simultaneously, we could have been forced to select only one of the five outputs to be fed to the optimizer. We avoided that by adding four of the five neural outputs into a single variable with weighted contributions for each (CSF was not a relevant variable to optimize), as represented by the following equation:

where:

$$y = \sum_i w_i \cdot n_i \quad (2)$$

y is combined neural output fed to the optimizer



***5. Simulation and optimization of the mechanical pulping of* Acer saccharum.**

w_i is weight associated with i[th] neural output
n_i is i[th] normalized neural output

In order to avoid scaling problems due to different engineering units for each dependent variable appearing in the above equation, they were all normalized in the interval [-1, +1] before being added. The neural network module automatically performs this normalization while it is computing its activations, as represented by the equation:
where:

$$o_i = \frac{\left(n_i + \frac{2 \cdot o_{i_{\max}}}{o_{i_{\max}} - o_{i_{\min}}} - 1 \right)}{\frac{2}{o_{i_{\max}} - o_{i_{\min}}}} \qquad (3)$$

n_i is i[th] normalized neural output
o_i is i[th] neural output (in engineering units)

This way, the weight multiplying each normalized dependent variable in *Eq. 2* serves the purpose of tuning the desired contribution of each output in the sum, without any relation to engineering units. Weighting all others to zero can optimize a single dependent variable. The engineering value of any variable can be traced back using the inverse relation:

$$n_i = \frac{2 \cdot o_i}{o_{i_{\max}} - o_{i_{\min}}} - \frac{2 \cdot o_{i_{\max}}}{o_{i_{\max}} - o_{i_{\min}}} + 1 \qquad (4)$$

The neural network module automatically performs this inverse mapping and carries out engineering units to its labeled output streams (Fig. 5).

Handling maximization

The optimizer module always performs minimization of the dependent variable. Maximization problems can be easily handled by appropriately changing the dependent variable's value. In this example, we wanted to minimize refining energy while maximizing optical and mechanical properties. Therefore, we modified the normalized neural output for brightness, tear index, and breaking length by premultiplying it with a reversal value of minus one.

Sharing variables

The refiner module, the neural network module, and the optimizer module all interact within the same simulation and simultaneously share some simulation variables. For example, the Na_2SO_3 and NaOH variables are coordinates computed by the optimizer and fed back to the neural network input which then feeds it to the impregnator's input streams. The refining energy output of the neural net feeds back to the refiner module for computation of the new fiber fractions at that energy. At the same time, refining energy is extracted from the neural network to be part of the weighted sum for input to the optimizer.

Simulation results

Table I shows a number of optimization results. The first column presents results using the downhill simplex method. All four output variables were given equal weight. As can be seen, the best combination found for temperature, NaOH, Na_2SO_3, and plate gap were respectively 70°C, 2%, 4%, and 0.3 mm. These values correspond to a cost of –0.7966 for the objective function. Recall that in weighted normalized space with four simultaneous dependent variables, -4 would correspond to the best possible output point while +4 would correspond to the worst one. The returned negative value of –0.7966 is therefore reasonable as all four optimized output variables are fighting each other at equal weight.

Other runs of the simulation in standard simplex mode always gave the same result, as did the simulated annealing version. This is explained by the fact that the landscape modeled by the neural network is rather smooth and "slants" downwards towards the coordinates 70°C, 2%, 4%, and 0.3 mm. These are all limiting

	ALL EQUAL WEIGHTS (downhill simplex and annealing)	SIMULATED ANNEALING Ten times more weight given to refining energy
Temperature (°C)	70.006	129.999
NaOH (%)	1.998	1.999
Na_2SO_3 (%)	3.999	3.995
Plate gap (mm)	0.300	0.499
Objective function	-0.7966	-3.985

I.* A. saccharum *mechanical pulping optimization results.

hyperplanes for the independent variables, corresponding to the range for which the neural network was trained. Consequently, the simplex shrinks and stops in that region. There is therefore a well-defined global minimum for this problem that is consistently found and returned by both algorithms.

Next we ran the simulation by providing more weight to the refining energy variable. As expected, the results returned in the second column of Table I show that the minimum lies at the highest possible values for the temperature (130°C) and plate gap (0.5 mm), both corresponding to limiting hyperplanes.

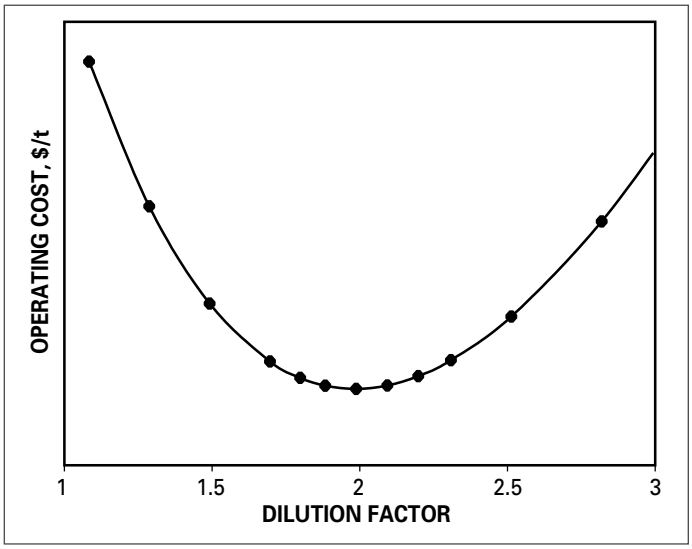
The example simulation, although quite complete, was instantaneous: every change in an input variable propagates instantaneously to output variables (there are no dynamics involved). The optimization module can still be used in dynamic simulations using a so-called "deadtime" parameter. It tells the optimization module to wait "deadtime" units of simulation time (specified in numbers of minutes) before it polls the simulation engine for the value of the objective function resulting from some coordinate changes. This simplistic method works well with dynamic simulations. The drawback is that the dynamics are not known to the optimizer. Because of that, whenever it throws in new coordinates corresponding to some simplex move, it must wait for the dynamics to settle before polling the new objective function value from such coordinates. This can add substantially to the optimization time.

To test the optimization module on a more challenging dynamic process, we tested it on a simulation of a brown stock washing operation. Optimal wash flows exist for given conditions [3]. Too much wash water causes high evaporation costs; too little results in chemical losses and higher bleaching costs.

The optimization module was inserted into a simulation where it suggested wash water flows, with the simulation totaling operating costs as the objective function. The optimization module successfully found the minimum represented in **Fig. 6** if the "deadtime" parameter was set sufficiently large, allowing the process to settle between successive simplex moves.

CONCLUSIONS

This paper discussed the implementation of an optimization module within a commercial simulation software system. One of its major characteristics is its flexibility. You can easily integrate the module into any existing simulation. All that is required is to specify the independent variables, the dependent variable to optimize, and the boundaries on the independent variables in order to define hyperplanes that must not be crossed by the solution engine (constrained optimization). When using a simulated annealing version of the downhill simplex method, a further fractional convergence parameter must be supplied which will influence the temperature decrease rate, and in turn, the speed of convergence and quality of the solution returned. This trade-off between quality and time is typical of any complex optimization problem and is typically problem-dependent.



6. Operating cost curve as a function of dilution factor.

Looking at the optimization problem in terms of its inherent difficulty, we have problems with few variables that have a unique and well-defined minimum on one end of the difficulty spectrum. At the other end, we have multivariable complex functional relationships where many local minimums that are in the same order of magnitude might be present. In this case, when using the downhill simplex method, which minimum is returned will depend on the randomly selected points of the initial simplex. You should therefore use different multiple reruns of the simulation to compare which returned minimum could possibly constitute an overall minimum. When using the simulated annealing version, the local minimum found has greater probability of being the global minimum, especially if the annealing parameter is decreased slowly.

In the test case on dynamic brown stock washing, the dynamics were such that it took 4000 min. of simulation time (about 1 min. in real time) for the simulation to stabilize given some changes in an input coordinate. This means that a simplex move could only be initiated at each 4000 min. of simulation time. A better approach would be to make the optimizer aware of the dynamics involved by providing it with a dynamic model in order to predict the outcome resulting from coordinate changes instead of waiting for it to happen. Needless to say that for simulations with many interacting dynamics, this model can become quite difficult to generalize and program.

Overall, the developed optimization module is a good improvement to the commercial simulator into which it was implemented. It provides very good insight on the effects that combinations of independent variable values have on some critical dependent variable to be optimized. When combined with the neural network module, true optimization of important quality parameters can be performed. **TJ**

LITERATURE CITED

1. Aguiar H.C.I.L, and Filho R.M.: "Modeling and optimization of pulp and paper processes using neural networks," *Computers in Chemical Engineering*, (22), pp. S981-S984, 1998.
2. Perez I., Lopez F., Ariza J.L.G., and Jemenez L.: "Characterization of paper sheets from olive tree wood pulp obtained by soda pulping," abstracted in *TAPPI J.*, 84(1): 94(2001); full text at <www.tappi.org>.
3. Wasik L.S., Mittet G.R., and Nelson D.G., *TAPPI J.*, 83(3): 94-101 (2000).
4. Bérubé P.R. and Hall E.R. , *Pulp Paper Can.*, 101(3): 54-57 (2000).
5. Laperrière, L. and Wasik L.S., "Modeling and simulation of pulp and paper quality characteristics using neural networks," abstracted in *Solutions!*, 84(10): 59(2001); full text at <www.tappi.org>.
6. Bazaraa M.S., Sherali H.D. and Shetty C.M.: *Nonlinear programming: theory and algorithms*, Second edition, John Wiley & Sons Inc., 1993.
7. Press W.H., Teukolsky S.A., Vetterling W.T., and Flannery B.P., *Numerical recipes in C*, Second edition, Cambridge University Press, 1997.
8. Nelder J.A. and Mead R.: *Computer Journal*, (7), 1965, pp.308-313.
9. Kirkpatrick S.: *Journal of Statistical Physics*, (34), 1984, pp. 975-986.
10. Anon., *CADSIM Plus DLM Programmer's Manual, Revision 1.1*, Aurel Systems Inc., 4th printing, March 1998.

Received: September 17, 2001
Accepted: January 21, 2002

This paper was presented at the 2001 TAPPI Coating Fundamentals Symposium. It is also published on TAPPI's web site (www.tappi.org) and summarized in the June *Solutions! for People, Processes and Paper* magazine (Vol. 85 No. 6).

INSIGHTS FROM THE AUTHORS

Mills often use simulation to help optimize their processes and products. However, they usually accomplish that manually by "playing" with the simulation, running it with various combinations of process variables until one seems to be optimal. Doing optimization this way does not ensure a systematic search nor an optimal solution. A mathematically robust optimization is only achieved if optimization algorithms take part of the simulation.

On the practical side, optimization should be performed on key variables, which constitute important quality characteristics of the process (optical, mechanical, etc.). These usually do not take part in the mass-energy balances that most simulators use. We dealt with this first obstacle to true process optimization by using neural models independent of the type of input-output variables being functionally related. The research reported here is a direct compliment of this earlier research. It provides the additional mathematics on top of the neural models to search for combinations of variables that optimize the end product's key characteristics.

A challenge we faced in this study was in having to fully integrate the optimization a commercial simulation software package. One crucial factor was the compatibility of the computations required by the optimization procedures with those already performed by the simulator. This lead us to algorithms that did not require first or higher order derivatives of the objective function. All they need is the local value of the objective function given some values of the independent variables. This is something that the simulator provides at each iteration, i.e. for every change in the independent variable at iteration "t-1", new values for the dependent variable(s) are obtained at iteration "t."

Due to the generic nature of the simulator in which it had to be implemented, the developed optimizer also had to be generic also. That meant it had to optimize functional relationships that could be non-linear, multivariable, constrained, and dynamic. Off-the-shelf algorithms were a good start, but we quickly discovered that many customizations had to be developed.

The power behind the simulated annealing method is quite surprising. When tuned with adequate parameters, this method has a very high probability of finding the global optimum of even complex functional relationships. It is very interesting to use this method and see it find the global optimum after failing to do so with other more traditional methods.

For now, the optimizer handles the process dynamics by waiting until the process stabilizes before polling the new objective function value from new combinations of independent variables. This might be avoided by giving the optimizer a predictive model of the process so it can predict the consequences of new combinations instead of waiting for them to happen.

— Luc Laperrière

Laperrière

Wasik

Laperrière is with Université du Québec à Trois-Rivières, Pulp and Paper Research Center, Trois-Rivières, Quebec, Canada; Wasik is with Aurel Systems Inc., Burbaby, British Columbia; Email Laperriere at luc_laperriere@uqtr.uquebec.ca, or Wasik at mail@aurel.bc.ca